

МИНОБРНАУКИ РОССИИ

**Федеральное государственное автономное образовательное
учреждение высшего образования
«Южный федеральный университет»**

**Институт математики, механики
и компьютерных наук им. И. И. Воровича**

Рахилова Екатерина Олеговна

**СОЗДАНИЕ РЕСУРСА ДЛЯ АВТОМАТИЧЕСКОГО
УПРАВЛЕНИЯ ДОКУМЕНТООБОРОТОМ В
ОРГАНИЗАЦИИ**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по направлению подготовки

**02.04.02 – Фундаментальная информатика и информационные технологии,
направленность программы**

«Разработка мобильных приложений и компьютерных игр»

Научный руководитель –
доц., к. ф.-м. н. Махно Виктория Викторовна

Рецензент –
к. ф.-м. н. Кузнецова Елена Михайловна

Допущено к защите:

руководитель

образовательной программы

Демяненко Я.М.

Ростов-на-Дону – 2022

**Задание на выпускную квалификационную работу
студента 2 курса магистратуры Рахиловой Е. О.**

Направление подготовки: 02.04.02 - Фундаментальная информатика и информационные технологии

Студент: Рахилова Е. О.

Научный руководитель: доцент кафедры информатики вычислительного эксперимента к. ф.-м. н. Махно В. В.

Год защиты: 2022

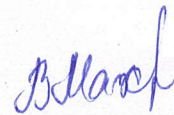
Тема работы: Создание ресурса для автоматического управления документооборотом в организации

Цель работы: Проанализировать разработку API, а также создать сервис, содержащий данные, необходимые пользователю для отображения в программах планирования задач.

Задачи работы:

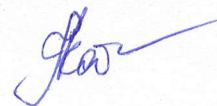
- Проанализировать функционал API;
- Реализовать программу на языке программирования Python, собирающую данные о документах из рабочей папки на компьютере;
- Настроить облачную платформу Heroku, для публикации API;
- Подключить реализованную программу к базе данных MySQL;
- Реализовать API, содержащее данные с полезной информацией необходимой пользователю для отображения в программах планирования задач;
- Опубликовать готовый сервиса по анализу документов на облачную платформу Heroku;
- Протестировать сервиса в созданном чат-боте по отслеживанию сроков сдачи документов;

Научный руководитель



Махно В. В.

Студент 2 курса магистратуры



Рахилова Е. О.

Руководитель программы



Демяненко Я.М.

23 сентября 2021 г.



СПРАВКА

Южный Федеральный Университет

о результатах проверки текстового документа
на наличие заимствований

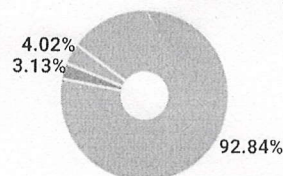
ПРОВЕРКА ВЫПОЛНЕНА В СИСТЕМЕ АНТИПЛАГИАТ.ВУЗ

Автор работы: Рахилова Екатерина Олеговна
Самоцитирование
рассчитано для: Рахилова Екатерина Олеговна
Название работы: СОЗДАНИЕ РЕСУРСА ДЛЯ АВТОМАТИЧЕСКОГО УПРАВЛЕНИЯ ДОКУМЕНТООБОРОТОМ В
ОРГАНИЗАЦИИ
Тип работы: Магистерская диссертация
Подразделение: ИММиКН им И.И.Воровича

РЕЗУЛЬТАТЫ

ЗАИМСТВОВАНИЯ	3.13%
ОРИГИНАЛЬНОСТЬ	92.84%
ЦИТИРОВАНИЯ	4.02%
САМОЦИТИРОВАНИЯ	0%

ДАТА ПОСЛЕДНЕЙ ПРОВЕРКИ: 05.06.2022



Модули поиска: ИПС Адилет; Библиография; Сводная коллекция ЭБС; Интернет Плюс; Сводная коллекция РГБ; Цитирование; Переводные заимствования (RuEn); Переводные заимствования по eLIBRARY.RU (EnRu); Переводные заимствования по Интернету (EnRu); Переводные заимствования издательства Wiley (RuEn); eLIBRARY.RU; СПС ГАРАНТ; Медицина; Диссертации НББ; Перефразирования по eLIBRARY.RU; Перефразирования по Интернету; Перефразирования по коллекции издательства Wiley; Патенты СССР, РФ, СНГ; СМИ России и СНГ; Модуль поиска "ЮФУ"; Шаблонные фразы; Кольцо вузов; Издательство Wiley; Переводные заимствования

Работу проверил: Махно Виктория Викторовна

ФИО проверяющего

Дата подписи:

Подпись проверяющего



Чтобы убедиться
в подлинности справки, используйте QR-код,
который содержит ссылку на отчет.

Ответ на вопрос, является ли обнаруженное заимствование
корректным, система оставляет на усмотрение проверяющего.
Предоставленная информация не подлежит использованию

Оглавление

Введение	2
Постановка задачи.....	3
1.Application Programming Interface (API).....	5
1.1 Существующие виды API	7
1.2 Протоколы и архитектуры API	8
2. Создание API для автоматизированного контроля сроков исполнения документов.	11
2.1 Используемые фреймворки и библиотеки.	11
2.2 Создание программы для сканирования документов	17
2.3 Подключение облачной платформы Heroku	22
2.4 Создание базы данных MySQL для хранения информации о документах.....	27
2.5 Создание API, используя фреймворк Flask	33
2.6 Тестирование созданного сервиса в чат-боте для контроля сроков сдачи документов	38
Заключение	1
Список литературы	1

Введение

В современном мире существует множество программ для планирования задач, которые очень помогают в повседневной жизни. Как правило, их функционал довольно примитивен. Ниже приведены стандартные возможности:

- Добавление/удаление задачи;
- Установка напоминания;
- Добавление приоритетов;
- Сохранение текущих задач (Необходимо, чтоб при обновлении программы все задачи сохранялись);

При заведении задач по документам с напоминанием о сроке сдачи в силу человеческого фактора какой-нибудь из документов окажется пропущенным, а это в свою очередь может повлечь за собой немало трудностей. Для решения данной проблемы в рамках выпускной квалификационной работы создан удобный сервис для сбора полезной информации, которая необходима пользователю. Данный сервис расширит функционал программ для планирования задач и еще больше поможет пользователям. Обмен данными между программами для планирования задач и созданным в рамках данной работы сервисом будет происходить по средствам API (программного интерфейса).

Постановка задачи

Целью данной выпускной магистерской работы является проведение анализа разработки API, а также создание сервиса, содержащего данные, необходимые пользователю для отображения в программах планирования задач.

Для достижения цели решены следующие задачи:

- Анализ функционала API;
- Реализация программы на языке программирования Python, собирающую данные о документах из рабочей папки на компьютере;
- Настройка облачной платформы Heroku, для публикации API;
- Подключение реализованной программы к базе данных MySQL;
- Реализация API, содержащее данные с полезной информацией необходимой пользователю для отображения в программах планирования задач;
- Публикация готового сервиса по анализу документов на облачную платформу Heroku;
- Тестирование сервиса в созданном чат-боте по отслеживанию сроков сдачи документов;

1. Application Programming Interface (API)

API - это набор функций, позволяющих приложениям получать доступ к данным и взаимодействовать с внешними программными компонентами, операционными системами или микросервисами. Прежде всего, необходимо разобраться, по какому принципу происходит обмен данными (рис. 1).

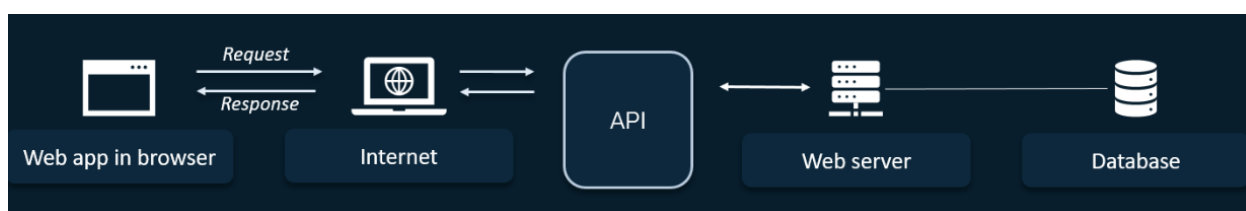


Рис. 1. Схема обмена данными

API состоит из двух частей:

- Техническая спецификация, описывающая возможности обмена данными между решениями, причем спецификация выполняется в виде запроса на обработку и протоколов доставки данных.
- Программный интерфейс, написанный в соответствии с представляющей его спецификацией.

Другими словами, API передает ответ пользователя системе и отправляет ответ системы обратно пользователю. Пользователь нажимает кнопку "подписаться на популярного писателя" в популярной социальной сети Твиттер; API сообщает сайту, что он планирует подписаться на другого объекта; сайт обновляет информацию, добавляет пользователю в подписки нового объекта и транслирует в ленту его популярные публикации.

Ранее, при определении понятия API, был упомянут термин «микросервисы». Однако API и микросервис это не одно и то же.

Микросервисы — это стиль или архитектура, которая разделяет функциональность внутри веб-приложения. В то время как API — это основа, с помощью которой разработчики взаимодействуют с веб-приложением.

API позволяет разработчику сделать определенный "запрос", чтобы отправить или получить информацию. Эта связь осуществляется с помощью текстового формата данных под названием "JSON".

JavaScript Object Notation (JSON) — это легкий и простой в разборе текстовый формат для обмена данными. Каждый файл JSON содержит коллекции пар имя/значение и упорядоченные списки значений. Поскольку это универсальные структуры данных, формат может быть использован с любым языком программирования (Листинг 1).

Значения ключа могут быть любыми:

- число
- строка
- массив
- другой объект

Листинг 1. Пример JSON файла

```
{
  "name": "Ivan Ivanov",
  "id": 7,
  "personal_data": {
    "age": 27,
    "username": "ivanov_ivan",
    "email": "ivanov@newbox.com",
  },
  "subscribers": 1500,
  "subscriptions": 200,
  "publications": 138,
  "is_banned": false,
}
```


1.1 Существующие виды API

С точки зрения политики безопасности, API могут быть частными, партнерскими и публичными. Рассмотрим каждый из них отдельно.

1. Частные API.

Частные API предназначены только для использования внутри предприятия для соединения систем и работы с данными. Например, частный API может соединять системы расчета заработной платы и управления персоналом организации.

2. Партнерские API.

Чаще всего партнерские API облегчения деятельности между предприятиями. Одна сторона формирует данные, а вторая приобретает доступ к этим данным у первой. Компания, создающая API, может контролировать, как используются их сервис и следить за тем, обеспечивают ли сторонние решения, использующие их API, достойный пользовательский опыт. Партнеры имеют четкие права и лицензии на доступ к таким API. По этой причине партнерские API обычно включают в себя достаточно надежные механизмы аутентификации, авторизации и безопасности.

3. Публичные API.

Публичный API является открытым и доступным для использования любым сторонним разработчиком или предприятием. Публичные API обычно предполагают умеренную аутентификацию и авторизацию. Предприятие также может попытаться монетизировать API, установив стоимость каждого вызова для использования публичного API.

1.2 Протоколы и архитектуры API

API обмениваются командами и данными, и это требует четких протоколов и архитектур — правил, структур и ограничений, которые регулируют работу API. Сегодня существует множество архитектурных стилей API и самые популярные из них это REST и SOAP. Их можно назвать "форматами", каждый из которых обладает уникальными характеристиками и компромиссами и используется для различных целей. Рассмотрим каждую спецификацию, чтобы выяснить, какая из них больше всего подходит для решения поставленных задач:

1. SOAP API

Simple Object Access Protocol (SOAP) — это протокол веб-коммуникаций, который используется для обмена информацией и структурированными данными через HTTP/HTTPS. В отличие от REST API, SOAP поддерживает только формат данных XML. В основном обмен данными в Интернете осуществляется через REST, но это не означает, что SOAP скоро уйдет в прошлое, поскольку он высоко стандартизирован, более безопасен и в некоторых случаях позволяет автоматизировать процесс.

XML или eXtensible Markup Language — это независимый от программного или аппаратного обеспечения инструмент для хранения и транспортировки данных. Он имеет набор правил для структурирования сообщений в виде удобных как для человека, так и для машины записей. Его структура похожа на HTML, но в языке XML нет predefined тегов, поэтому автор должен определить как теги, так и структуру документа (Листинг 2).

Листинг 2. Пример структуры XML файла

```
<?xml version="1.0"?>
<catalog>
  <book id="book1">
    <author>Ivanov, Ivan</author>
    <title>About XML Format</title>
    <price>10.15</price>
    <genre>Computer</genre>
    <publish_date>2020-10-01</publish_date>
    <description>How to create an XML file</description>
  </book>
</catalog>
```

2. REST API

Архитектура репрезентативной передачи состояния (REST) является, пожалуй, самым популярным подходом к созданию API. REST опирается на подход клиент/сервер, который разделяет внешний и внутренний интерфейс API, и обеспечивает значительную гибкость в разработке и реализации. REST не является статичным, что означает, что API не хранит данные или статус между запросами. API REST, обычно называемые "RESTful API", также могут взаимодействовать напрямую или работать через промежуточные системы, такие как API-шлюзы и распределители нагрузки. Эти API используют HTTP-запросы для работы с ресурсами.

Существует четыре основных метода запроса:

- GET - Сбор информации (Получение всех данных).
- PUT - Обновление части данных (Обновление цен на товары)
- POST – Добавление нового элемента (Создание новой категории продукта).
- DELETE – Удаление элемента (Удаление записи в блоге).

В отличие от SOAP, REST не ограничивается использованием XML для предоставления ответа. Мы можем получить информацию в различных форматах, таких как JSON, XML, CSV, простой текст, HTML, которые легче

разбираются в зависимости от языка, который вы используете для своего приложения.

Способность поддерживать множество форматов для хранения и обмена данными - одна из причин, по которой была выбрана архитектура REST в качестве создания API для нашей программы.

2. Создание API для автоматизированного контроля сроков исполнения документов.

Для выполнения одной из поставленных целей выпускной работы, был выбран язык программирования Python.

Python — это язык программирования общего назначения, интерпретируемый и высокоуровневый. Он ориентирован на удобочитаемость и простой синтаксис. Python предлагает все функциональные возможности, которые могут понадобиться для решения задач программирования.

Для редактирования исходного кода программы была выбрана программа Atom.

Atom позволяет разработчикам продуктивно использовать все возможности текстового редактора, никогда не прикасаясь к конфигурационным файлам. Только что загруженная версия поставляется с восемью темами для подсветки синтаксиса и четырьмя UI: двумя светлыми и двумя темными. Но если ни одна из предустановленных тем пользователя не заинтересовала, Atom позволяет легко и быстро установить пользовательские темы, созданные сторонними разработчиками, или создать их самостоятельно.

2.1 Используемые фреймворки и библиотеки.

Одной из основных фреймворков Python, используемых в разработке API, является Flask.

Flask — это так называемый фреймворк WSGI. В переводе означает "интерфейс шлюза веб-сервера". По сути, это способ для веб-серверов

передавать запросы веб-приложениям или фреймворкам. Flask опирается на внешнюю библиотеку WSGI, а также на механизм шаблонизатора Jinja2. Перечислим несколько преимуществ и недостатков Flask.

Преимущества Flask:

1. Масштабируемый.

Статус Flask как микрофреймворка и его размер означают, что можно использовать его для невероятно быстрого роста такого технологического проекта, как веб-приложение. Если необходимо создать приложение, которое начинается с малого, но имеет потенциал для быстрого роста в различных направлениях, которые еще не до конца проработаны, то это идеальный выбор. Простота использования и небольшое количество зависимостей позволяют ему работать без сбоев даже при увеличении масштаба.

2. Гибкий.

Это основная особенность Flask и одно из его самых больших преимуществ. Перефразируя один из принципов Zen of Python, можно сказать, что простота лучше сложности, потому что ее можно легко переставлять и перемещать. Это не только полезно с точки зрения того, что ваш проект может легко двигаться в другом направлении, но и гарантирует, что структура не разрушится при изменении какой-либо части. Минимальный набор компонентов Flask и его пригодность для разработки небольших веб-приложений означает, что он даже более гибок, чем сам Django.

3. Прост в навигации.

Как и в случае с Django, возможность легко ориентироваться в нем является ключевым фактором, позволяющим веб-разработчикам сосредоточиться на быстром кодировании, не увязая в нем. По своей сути микрофреймворк прост для понимания веб-разработчиков, что не только экономит их время и усилия, но и дает им больше контроля над своим кодом и возможными возможностями.

4. Легковесный.

Когда мы используем этот термин по отношению к инструменту или фреймворку, мы говорим о его конструкции - в нем мало составных частей, которые нужно собирать и пересобирать, и он не зависит от большого количества расширений для функционирования. Такая конструкция дает веб-разработчикам определенный уровень контроля. Flask также поддерживает модульное программирование, то есть его функциональность может быть разделена на несколько взаимозаменяемых модулей. Каждый модуль действует как независимый строительный блок, который может выполнять одну часть функциональности. В совокупности это означает, что все составные части структуры сами по себе являются гибкими, подвижными и тестируемыми.

5. Документация.

Следуя теории создателя о том, что "хороший дизайн документации заставляет вас действительно писать документацию", пользователи Flask найдут большое количество примеров и советов, расположенных в

структурированном виде. Это стимулирует разработчиков к использованию фреймворка, поскольку они могут легко ознакомиться с различными аспектами и возможностями инструмента. Документацию по Flask можно найти на официальном сайте.

Недостатки Flask:

1. Недостаточно инструментов разработки.

Легкость этого микрофреймворка неизбежно имеет некоторые недостатки. Главным из них является то, что в отличие от Django, Flask не имеет большого набора инструментов. Это означает, что разработчикам придется вручную добавлять расширения, такие как библиотеки. И если вы добавите огромное количество расширений, это может привести к замедлению работы самого приложения из-за множества запросов.

2. Сложность ознакомления с большим приложением, написанным на Flask.

Из-за того, что разработка веб-приложения с использованием Flask может принимать различные повороты и изгибы, веб-разработчику, пришедшему в проект на середине пути, будет трудно понять, как он был разработан. Модульная природа микрофреймворка, о которой мы говорили ранее, может снова преследовать программистов, которым придется знакомиться с каждой составной частью.

Взвесив все преимущества и недостатки данного микрофреймворка, можно сделать вывод, что с Flask легко начать работу, потому что он не требует больших усилий для обучения. Кроме того, он очень понятен, что повышает читабельность.

Для парсинга дат сдачи в документе в формате .docx был использован python пакет под названием «docx2txt». Данный пакет позволяет извлекать текст и изображения из документов Word. Мы можем прочитать документ, используя метод пакета под названием process, который принимает имя файла в качестве входных данных. Обычный текст, элементы списка, текст гиперссылок и текст таблицы будут возвращены в виде одной строки.

А для документов в формате .pdf была использована библиотека «PyPDF2». Используя данную библиотеку, можно совершать множество различных типов операций с PDF.

Используя этот пакет, можно:

- Извлекать нужные данные из файла
- Менять ориентацию страниц
- Объединять файлы
- Разделять файлы
- Добавлять водяные знаки
- Шифровать файл

Также были задействованы такие python модули, как «OS» и «RE (regular expression)».

Модуль «OS» в Python является частью стандартной библиотеки языка программирования и поставляется с множеством различных функций, которые позволяют разработчикам взаимодействовать с операционной системой (Macintosh, Windows, Linux), над которой они в данный момент работают.

Модуль «RE» как раз предоставляет набор функций, позволяющих искать совпадения в строке по регулярным выражениям, в которых прописана последовательность искомых символов или шаблонов.

Для создания интерфейса программы использовалась библиотека «PySimpleGUI».

На данный момент существует 4 активно разрабатываемых и поддерживаемых "порта" PySimpleGUI. К ним относятся:

- tkinter - Полностью завершен
- Qt, использующий Pyside2 - Альфа стадия. Не все функции для всех элементов сделаны
- WxPython - стадия разработки, предварительный релиз. Не все элементы завершены.
- Remi (поддержка веб-браузера) - Стадия разработки, пререлиз.

Порт tkinter, является единственной на 100% завершенной версией PySimpleGUI, остальные 3 порта имеют много функциональности и активно используются значительной частью установленных программ.

Также для удобства обращения к папке с документами был задействован модуль «pathlib»

Этот модуль предлагает классы, представляющие пути файловой системы с семантикой, подходящей для различных операционных систем. Классы путей делятся на чистые пути, которые обеспечивают чисто вычислительные операции без ввода/вывода, и конкретные пути, которые наследуются от чистых путей, но также обеспечивают операции ввода/вывода.

Для возможности использования базы данных в программе использовался модуль mysql.connector.

Данный модуль позволяет программам на Python получать доступ к базам данных MySQL, используя API, соответствующий спецификации Python Database API Specification v2.0 (PEP 249). Преимущества данного модуля в том, что он написан на чистом Python, и является самодостаточным

для выполнения запросов к базе данных через Python. Также `mysql.connector` официальный поддерживаемый Oracle драйвер для работы с MySQL и Python. Он совместим с Python 3 и активно поддерживается (рис. 2).

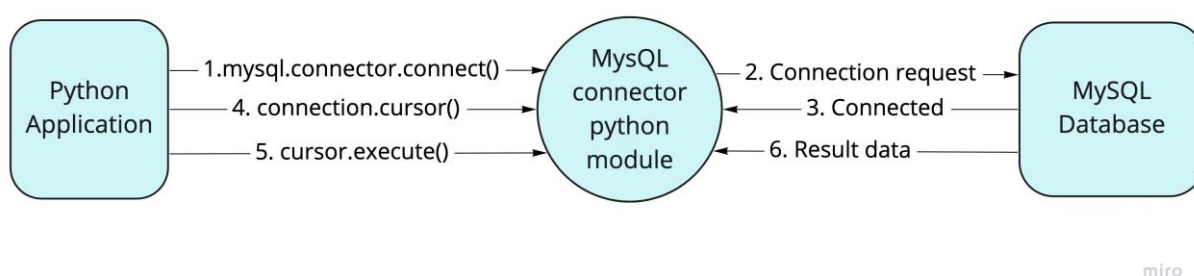


Рис. 2. Схема работы модуля `mysql.connector`

2.2 Создание программы для сканирования документов

Перед тем, как непосредственно начать писать программу на языке Python, при помощи командной строки были установлены пакеты, библиотеки и микрофреймворк, описанные ранее. Для простоты установки каждого из инструментов, в Python существует пакетный менеджер под названием Pip. Данный пакетный менеджер создан для того, чтобы устанавливать сторонние пакеты и библиотеки, которые изначально не входят в Python.

Перечислим команды, которые использовались для установки сторонних пакетов, модулей и библиотек:

- Для docx2txt — «`pip install docx2txt`»
- Для PyPDF2 — «`pip install PyPDF2`»
- Для Flask — «`pip install flask`» и «`pip install flask-restful`»

- Для MySQL — «pip install mysql-connector-python»
- Для PySimpleGUI — «pip install pysimplegui»

После установки был создан файл под названием «document-service» с расширением «.PY».

Для поддержания актуальности версий программы была настроена связь нашего локального проекта с Github. Для этого создается репозиторий в личном кабинете. После создания репозитория, предоставляется возможность «клонировать» его на собственный компьютер используя команду «git clone github.com/ {имя_пользователя}/ document-service.git».

После клонирования репозитория автоматически создается папка, в которую нам нужно положить Python файл, который мы создали ранее.

Также к локальному проекту был настроен клиент SourceTree для удобства взаимодействия с GitHub.

Sourcetree — это мощный клиент для взаимодействия с Git, который Atlassian бесплатно предоставляет как для Windows, так и для Mac. Использование Sourcetree позволяет упростить взаимодействие с кодом, включая визуализацию, улучшая способ управления вашими репозиториями. Благодаря доступным подробным диаграммам ветвления можно легко следить за каждым обновлением, которое было сделано.

После добавления репозитория, данный клиент покажет, что в нем произошли изменения, поскольку ранее в папку был перенесен файл под названием «document-service».

Для отправки внесенных изменений в удаленный репозиторий, необходимо перенести изменения из состояния «Unstaged» в «Staged». Далее нажимаем на кнопку «Commit», пишем комментарий о том, что было сделано и нажимаем на кнопку «Push».

Теперь, когда настроена связь с Github, перейдем непосредственно к разработке программы.

В начале файла пропишем все пакеты и библиотеки, которые мы установили ранее (Листинг 3).

Листинг 3. Подключение установленных пакетов и библиотек

```
from flask import Flask
from flask_restful import Api, Resource, reqparse
import docx2txt
from pathlib import Path
import mysql.connector
import PySimpleGUI as sg
import os
import re
```

Далее создадим функцию, которая будет принимать в себя список документов, которые в данный момент находятся в рабочей папке, а также пропишем список правил, по которому будет происходить фильтрация текста (Листинг 4).

Листинг 4. Функция files_enum() для анализа документов в папке

```
rules = ['срок сдачи до', 'выполнить до', 'подготовить ответ к',
        'выполнить до', 'сообщить до', 'согласовать до']

def files_enum(folder):
    files = os.listdir(folder)
    local_files = []

    for document in files:
        date = read_docs(str(folder) + '/' + document)
        obj = {
            'name': document,
            'date': date,
        }
        local_files.append(obj)
    return local_files

if __name__ == '__main__':
    files_enum(Path(__file__).parent/'university-documents')
```

Для определения даты сдачи документа была создана функция `read_docs()`, которая принимает в качестве аргумента путь к файлу (Листинг 5).

Листинг 5. Функция для получения даты сдачи

```
def read_docs(path):
    filename, file_extension = os.path.splitext(path)
    date = ''
    if file_extension == '.docx':
        date = parse_docx(path)
    if file_extension == '.txt':
        date = parse_txt(path)
    return date
```

Далее необходимо под каждый формат документа написать свою функцию парсинга данных.

Для формата TXT (Листинг 6):

Листинг 6. Функция для парсинга файла формата TXT

```
def parse_txt(path):
    text = open(path).read()
    date = ''
    for rule in rules: # Перебор правил
        if rule in text:
            date = text[text.index(rule):]
            date = str(re.search(r'\d\d.\d\d.\d\d\d\d',
date).group())
    return date
```

Для формата DOCX (Листинг 7):

Листинг 7. Функция для парсинга файла формата DOCX

```
def parse_docx(path):
    text = docx2txt.process(path)
    date = ''
    for rule in rules:
        if re.search(rule, text):
            date = text[text.index(rule):]
            date = str(re.search(r'\d\d.\d\d.\d\d\d\d',
date).group())
    return date
```

Для поиска даты в тексте использовался модуль «RE» для создания регулярных выражений. Функция `re.search()` будет искать шаблон регулярного выражения и возвращать первое вхождение. В отличие от функции Python `re.match()`, она проверяет все строки входной строки. Функция Python `re.search()` возвращает объект `match`, если шаблон найден, и `"null"`, если шаблон не найден.

Перед созданием интерфейса программы, используя PySimpleGUI, прежде всего необходимо создать «layout» (макет). Макет содержит в себе список списков. Каждый внутренний список определяет одну строку. Каждая строка состоит из элементов пользовательского интерфейса, которые отображаются на одной строке (Листинг 8).

Листинг 8. Создание макета для интерфейса

```
layout = [
    [sg.Text('Введите электронную почту SFEDU'),
     sg.InputText()],
    [sg.Text('Введите user ID от телеграма'), sg.InputText()],
    [sg.Text('Введите chat ID от телеграма'), sg.InputText()],
    [sg.Text('(чтобы узнать свой id необходимо зайти в
приложение телеграм и вызвать бота "Get my ID")')],
    [sg.Submit(), sg.Cancel()]
]

window = sg.Window('Document Service', layout)
```

Используемые элементы пользовательского интерфейса:

- `sg.Text` — для отображения постоянного текста;
- `sg.InputText` — для приема ввода во время выполнения программы;
- `sg.Submit()` — кнопка, для сохранения введенных данных;
- `sg.Cancel()` — кнопка, для отмены введенных данных;
- `sg.Window()` — используется для создания окна с заголовком "Document Service" и указанным макетом.

Вывод интерфейса выглядит следующим образом (рис.3):

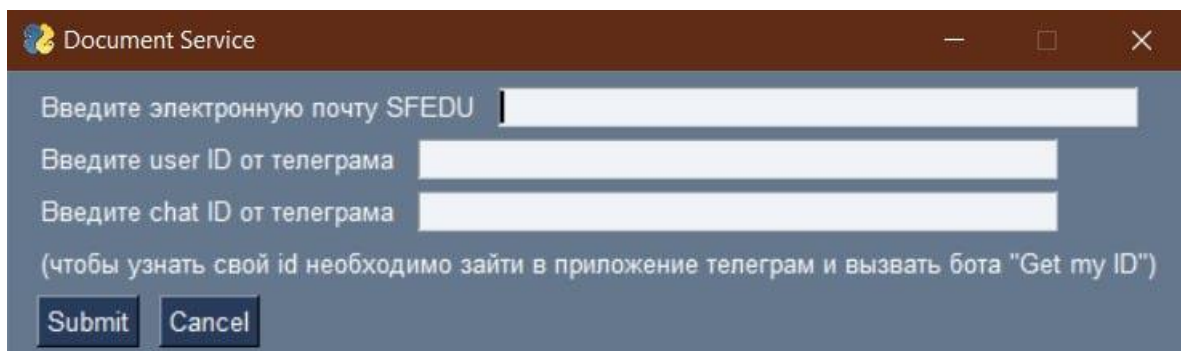


Рис.3. Внешний вид созданного интерфейса программы

Непосредственно для вызова данного интерфейса и считывания введенных данных, дополним наш код программы (Листинг 9).

Листинг 9. Вызов созданного интерфейса в программе

```
if name == 'main':  
    event, values = window.read()  
    if event in (None, 'Submit', 'Exit', 'Cancel'):  
        window.close()  
    email = values[0]  
    telegram_id = [values[1], values[2]]
```

В приведенном выше коде информация о событиях нажатия кнопки или закрытия окна и введенные данные получаются с помощью функции `window.read()`. При нажатии на кнопку закрытия окна происходит событие `sg.WINDOW_CLOSED`, в результате которого элемент управления выходит из цикла событий и окно закрывается с помощью функции `close()`.

Для передачи полученной информации из интерфейса в будущую базу данных, созданы переменные «email» и «telegram_id».

2.3 Подключение облачной платформы Heroku

В качестве сервиса для хранения API был выбрана Heroku. Heroku — это облачная PaaS (Platform as a Service) платформа, популярность которой

выросла в последние годы. Heroku настолько проста в использовании, что является лучшим выбором для многих проектов по разработке.

Платформа как услуга (PaaS) — это полноценная, масштабируемая среда разработки и развертывания, которая предоставляется по подписке. Основное преимущество модели PaaS заключается в том, что она позволяет пользователям получить доступ к аппаратному и программному обеспечению, которое можно использовать для разработки и запуска приложений без необходимости приобретения, установки и обслуживания инфраструктуры. Аналогичным образом, организация может использовать PaaS для расширения или изменения архитектуры существующих приложений в облаке.

Также существуют еще две модели облачных вычислений: SaaS (Software-as-a-Service) и IaaS (Infrastructure-as-a-Service). IaaS относится к инфраструктуре облачных вычислений - серверам, хранилищам и так далее - управляемая поставщиком облака, в то время как SaaS относится к полным приложениям, которые размещаются в облаке и обслуживаются поставщиком SaaS.

Облачные вычисления — это информационно-технологическая парадигма. Это модель для обеспечения повсеместного доступа к общим ресурсам, таким как компьютерные сети, серверы, хранилища, приложения и сервисы.

Приложения, запускаемые на Heroku, обычно имеют уникальные доменные имена, которые используются для маршрутизации HTTP-запросов к нужному контейнеру. Приложения как сервисы используют контейнеры приложений (Рис. 4).

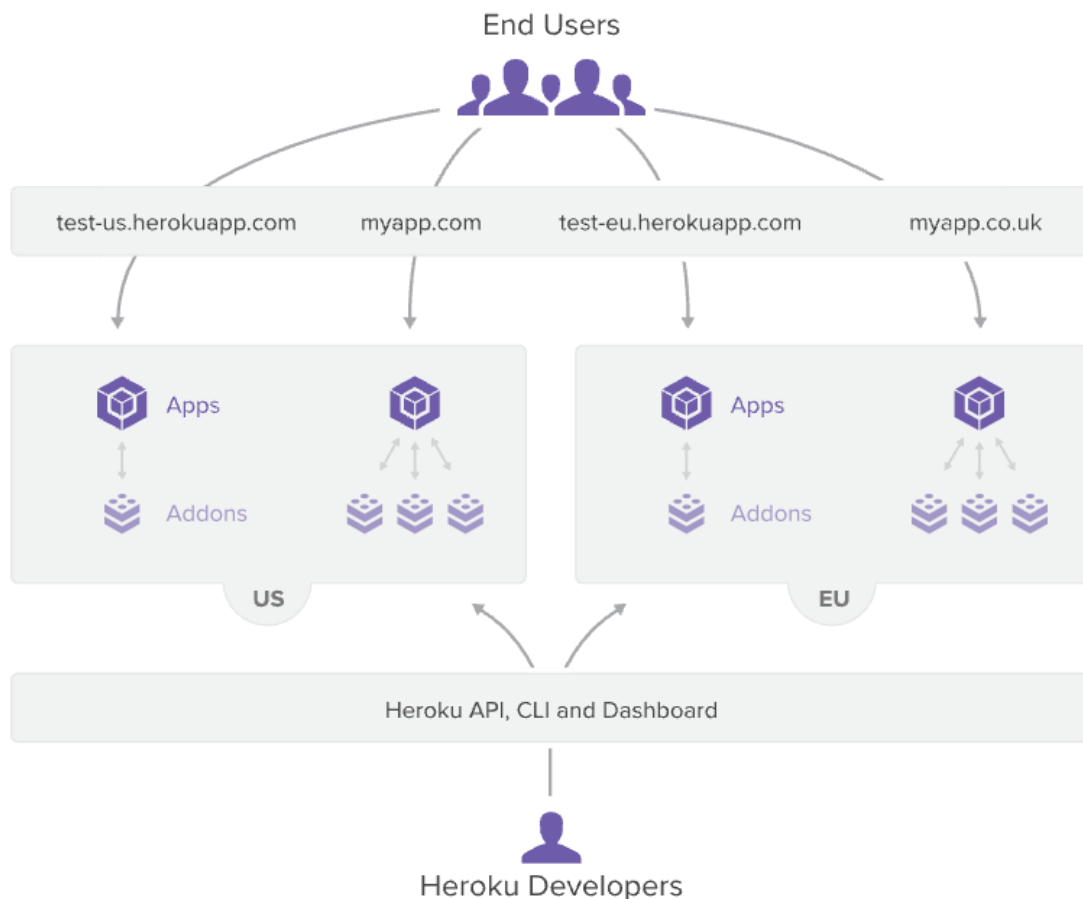


Рис.4 Принцип работы облачной платформы Heroku

Контейнеры предназначены для упаковки и запуска сервисов. Каждый из контейнеров приложений представляет собой интеллектуальный контейнер на надежной, полностью управляемой среде выполнения. Контейнеры приложений - называемые "dyno (дино или диносы)" в контексте платформы Heroku - распределены по "dyno-сетке". Она состоит из нескольких серверов. Менеджер Dyno поддерживает и управляет созданными dynos.

Heroku поддерживает несколько языков программирования, которые используются в качестве модели развертывания веб-приложений. Сейчас данная платформа поддерживает множество языков программирования, таких как: Java, Node.js, Python и так далее.

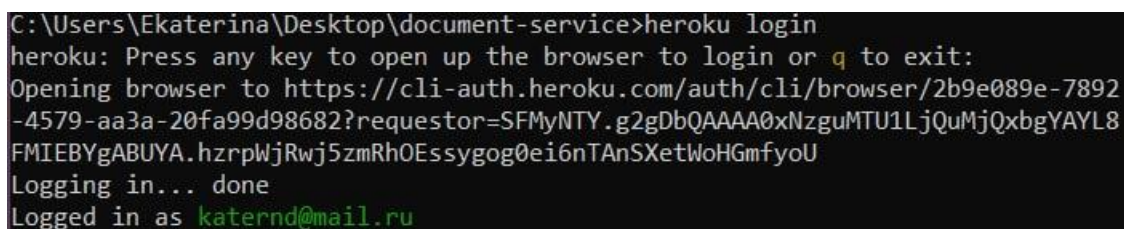
Полиморфизм и масштабируемость — это причины, по которым Heroku рассматривается как предпочтительная платформа для публикации сервиса.

Также отметим, что проекты, созданные в Heroku, привязаны к репозиториям в GitHub. Интеграция Heroku с GitHub обеспечивает автоматическую сборку и развертывание последней версии кода.

GitHub — это надежное хранилище исходного кода, поэтому интеграция Heroku с GitHub и другими инструментами помогает разработчикам оптимизировать свои усилия и экономить время и деньги заинтересованных сторон в ходе реализации проектов разработки.

Для подключения облачной платформы Heroku к созданной ранее программе, необходимо на основном сайте платформы скачать установочный файл в зависимости от операционной системы. В данном случае загружался установщик для Windows с 64-битной оперативной системой.

После установки используем команду «heroku login» для входа в Heroku CLI (рис. 5):



```
C:\Users\Ekaterina\Desktop\document-service>heroku login
heroku: Press any key to open up the browser to login or q to exit:
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/2b9e089e-7892
-4579-aa3a-20fa99d98682?requestor=SFMyNTY.g2gDbQAAAA0xNzguMTU1LjQuMjQxbgYAYL8
FMIEBYgABUYA.hzrpWjRwj5zmRhOEssygog0ei6nTAnSXetWoHGmfyoU
Logging in... done
Logged in as katernd@mail.ru
```

Рис.5 Авторизация в Heroku

Данная авторизация необходима для корректной работы команд Heroku и Git.

Перед выгрузкой программы на облачную платформу Heroku в папке проекта были созданы следующие необходимые файлы:

- requirements.txt — файл, который определяет, какие модули используются в проекте (Листинг 10)

Листинг 10. Файл requirements.txt

```
Flask==2.1.2
Flask-RESTful==0.3.9
gunicorn==20.1.0
docx2txt==0.8
pathlib==1.0.1
mysql.connector==2.2.9
```

- Procfile — определяет команды, выполняемые приложением при запуске.

После создания вышеупомянутых файлов можно приступить к выгрузке программы на платформу. Для достижения этой цели, выполнены следующие команды:

1. «heroku create sfedu-document-service --buildpack heroku/python»
2. «git init»
3. «git add .»
4. «git commit -m “Add new app” »
5. «git push heroku master»

Результат выполненных команд можно увидеть непосредственно на сайте платформы Heroku (рис. 6).

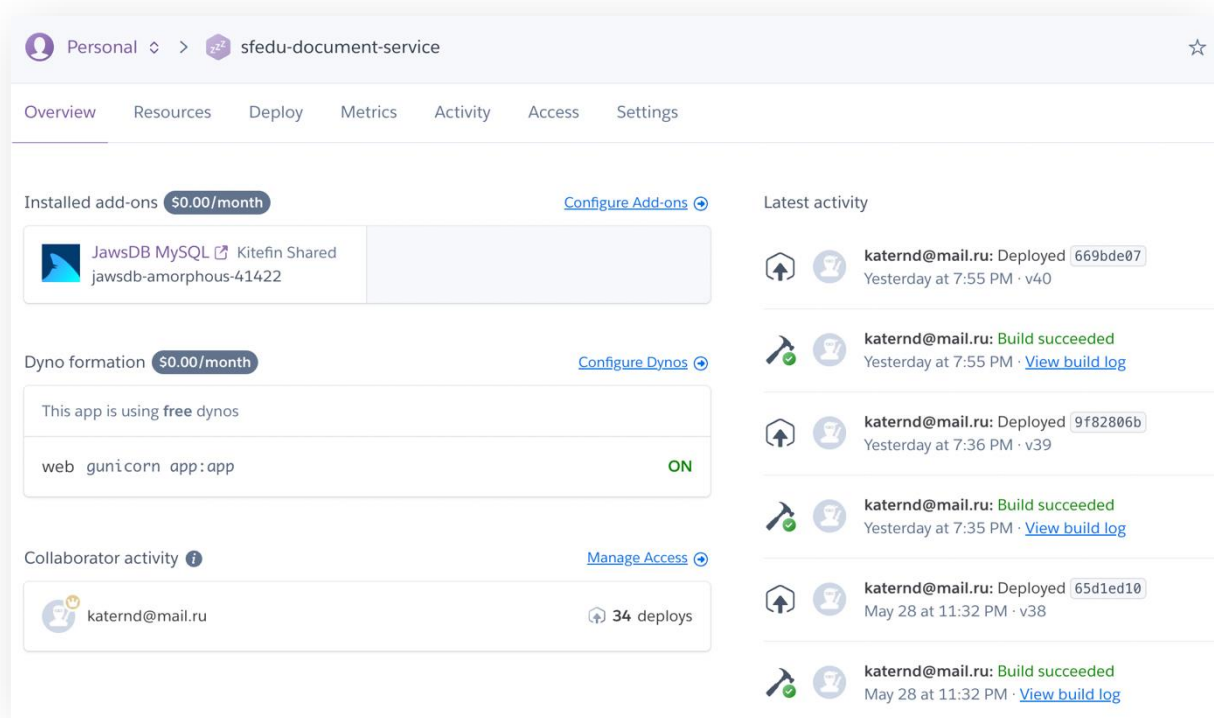


Рис. 6. Результат публикации программы на платформу Heroku

2.4 Создание базы данных MySQL для хранения информации о документах

В качестве хранения информации об отсканированных документах была выбрана система хранения баз данных MySQL.

MySQL — это РСУБД (расшифровывается, как реляционная система управления базами данных), реализующая SQL. Первоначально она была разработана для использования с базами данных малого и среднего размера, но теперь она может обрабатывать даже очень большие объемы хранимых данных. MySQL написана на языке C/C++ и в основном соответствует стандарту SQL. Однако в него добавлено множество расширений, а скорость и надежность важнее идеального соответствия. Более подробное

обсуждение соответствия MySQL и SQL можно найти в документации MySQL по стандартам соответствия.

Данная свободная реляционная система управления базами данных была выбрана по следующим причинам:

- Поддерживаются очень большие базы данных, хранящие до сотен миллионов записей и до 64 индексов на таблицу.
- MySQL доступен в виде библиотеки, которая может быть встроена в отдельные приложения.
- Позволяет осуществлять репликацию и резервирование данных для повышения надежности.
- Данная система известна своей высокой производительностью благодаря таким функциям, как оптимизация библиотек классов, сжатие, распределение памяти и хэш-таблицы. Она поддерживает многопоточное исполнение процессов ядра для более эффективной работы на системах с несколькими процессорами.
- MySQL предоставляет расширенные средства безопасности, включая шифрование всех паролей.
- Работает со многими различными компиляторами и на многих платформах и разработан для переносимости между системами. Клиентские программы могут быть написаны на многих языках.

Поскольку в рамках данной задачи задействована облачная платформа Heroku, мы можем воспользоваться встроенными расширениями данной платформы. Для упрощения создания базы данных, было выбрано расширение JawsJB.

JawsDB — это поставщик услуг базы данных как сервиса (DBaaS), предоставляющий полностью функциональную, полностью управляемую реляционную базу данных для использования в вашем приложении. Вместо того чтобы заниматься хостингом, настройкой, исправлением и управлением базой данных, JawsDB обеспечивает доставку и управление реляционной базой данных в облаке (рис. 7).

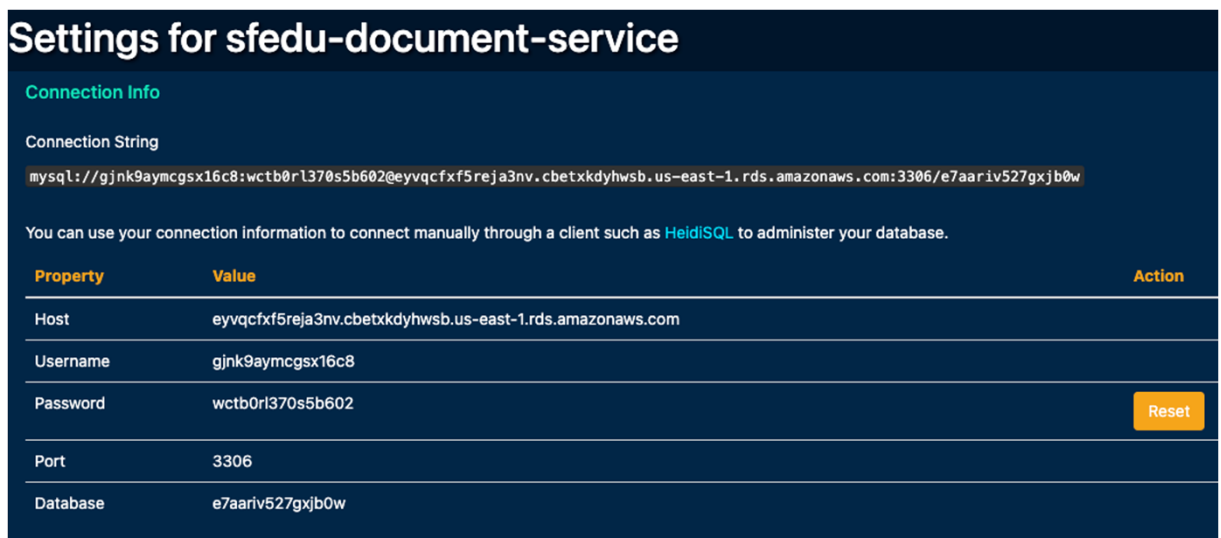


рис. 7. Подключенное расширение JawsDB

Для работы с базой данных была выбрана Интегрированная среда разработки под названием dbForge Studio.

dbForge Studio — это универсальная IDE для разработки, управления и администрирования баз данных MySQL и MariaDB. Это полноценный графический интерфейс MySQL, который помогает создавать и выполнять запросы, разрабатывать и отлаживать хранимые процедуры, автоматизировать управление объектами базы данных, сравнивать и синхронизировать базы данных, анализировать данные таблиц и многое другое. Богатые функциональные возможности предоставляются в интуитивно понятном интерфейсе.

Перед началом использования базы необходимо создать таблицу и ее ключи.

Для этого создадим простой запрос на создание таблицы «documents» со следующими ключами (рис. 8):

- `id` — номер документа (формируется автоматически);
- `name` — название документа;
- `visible` — видимость документа (опциональный ключ. Добавлен для удобства сервисов планирования задач)
- `email` — почта пользователя;
- `userID` — `id` пользователя в мессенджере «Telegram» (опциональный ключ. Добавлен для возможности подключения Telegram-бота);

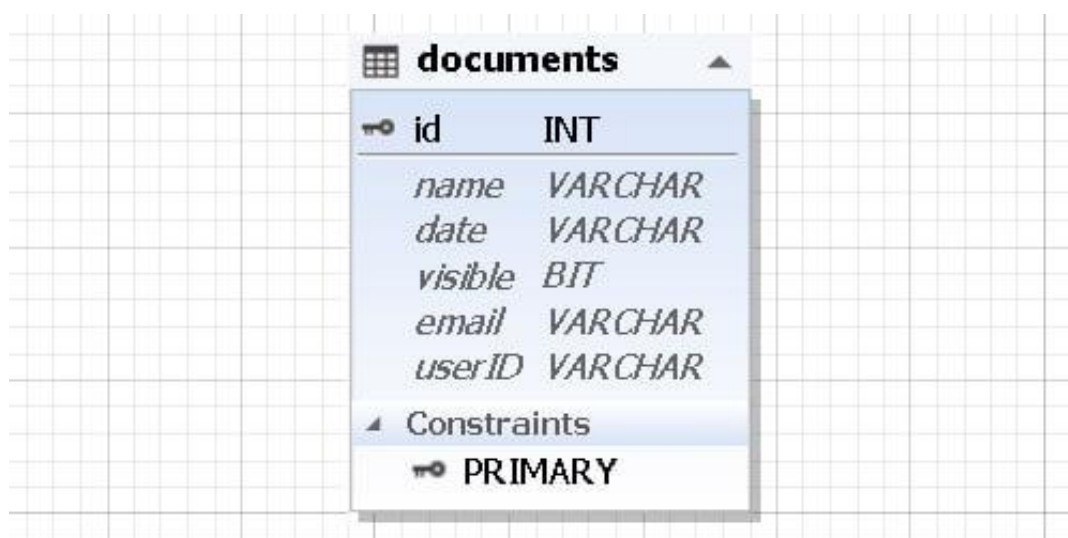


Рис. 8. Создание новой таблицы в базе данных MySQL

Для того, чтобы настроить взаимодействие написанной программы с созданной базой данных MySQL, был создан объект соединения, используя метод `mysql.connector.connect()`, который принимает в себя следующие параметры: имя пользователя, пароль, хост (необязательно, по умолчанию: `localhost`) и базу данных (необязательно). Затем был создан объект курсора,

который вызывает метод `cursor()` для объекта подключения, созданного ранее .

Чтобы выполнять запросы к уже подключенной базе данных, достаточно вызвать метод `execute()` и написать нужный запрос к таблице (Листинг 11).

Листинг 11. Подключение БД к программе сканирования документов

```
mydb = mysql.connector.connect(  
    host="eyvqcfx5reja3nv.cbetxkdyhwsb.us-east-  
1.rds.amazonaws.com",  
    user="gjnk9aymcgsx16c8",  
    password="wctb0rl370s5b602",  
    database="e7aariv527gxjb0w"  
)  
mycursor = mydb.cursor()  
mycursor.execute(f"""select name, date from documents""")
```

На данном этапе потребуется изменить созданную программу, чтобы при сканировании документов данные отправлялись в базу данных. Дополним уже существующую функцию `files_enum()` следующими строчками кода (Листинг 12):

Листинг 12. Добавление запросов к базе данных в функции `files_enum()`

```
mycursor = mydb.cursor()  
mycursor.execute(f"""select name, date from documents""")  
db_files = mycursor.fetchall()  
db_files_list = []  
  
for file in db_files:  
    db_files_list.append({'name':file[0], 'date':file[1]})  
copy_db_list = db_files_list  
  
for file in local_files:  
    if file in db_files_list:  
        copy_db_list.remove(file)  
  
    else:  
        mycursor.execute(f"""  
  
        INSERT INTO documents (name, date, visible, email,  
        userID)  
        VALUES ('{file['name']}', '{file['date']}', '{True}',  
        '{email}', '{telegram_id}');""")
```

```

mydb.commit()

for file in copy_db_list:
    mycursor.execute(f"""
        delete from documents
        where name='{file['name']}'""")

mydb.commit()

```

В данной функции также обработаны случаи, когда из рабочей папки удаляется файл и его также нужно удалить из таблицы, чтобы избежать дублей и не нужных данных в базе.

Также для удобства обращения к базе данных была создана общая функция, результатом выполнения которой является отформатированная и структурированная информации из базы (Листинг 13).

Листинг 13. Функция получение данных из базы

```

def get_database_data():
    mycursor = mydb.cursor()
    mycursor.execute(f"""select * from documents""")
    db_files = mycursor.fetchall()
    files_list = []

    for file in db_files:
        files_list.append({
            'id': file[0],
            'name': file[1],
            'date': file[2],
            'visible': file[3],
            'email': file[4],
            'telegramId': file[5],
        })
    return files_list

```

После запроса «SELECT» для удобства работы полученные данные формируются в список объектов.

2.5 Создание API, используя фреймворк Flask

Объект Flask реализует приложение WSGI и действует как центральный объект. Ему передается имя модуля или пакета приложения. После создания он будет действовать как центральный реестр для функций просмотра, правил URL, конфигурации шаблона и многого другого.

Также для создания API потребуется использовать расширение Flask под названием Flask- RESTful.

Flask-RESTful — это расширение для Flask, которое добавляет поддержку для быстрого создания REST API. Это легкая абстракция, которая работает с существующими ORM/библиотеками. Flask-RESTful поддерживает передовые методы с минимальной настройкой.

Flask-RESTful предоставляет базовый класс «Resource» и «Api». Resource может определять маршрутизацию для одного или нескольких методов HTTP для заданного URL-адреса. Класс Api нужен непосредственного для того, чтобы создать API (Листинг 14).

Листинг 14. Создание API, используя Flask

```
app = Flask(__name__)
api = Api(app)

api.add_resource(Documents, "/document-service", "/document-
service/", "/document-service/<int:id>")

if __name__ == '__main__':
    app.run(debug=True, use_reloader=False)
```

Метод `add_resource()` регистрирует маршруты во фреймворке, используя заданную конечную точку. Если конечная точка не указана, то Flask-RESTful генерирует ее за вас из имени класса.

В качестве параметров, данный метод может принимать в себя:

- `resource` (`Type[Resource]`) – имя класса ресурса
- `urls` (строка) – один или несколько url-маршрутов для поиска ресурса, применяются стандартные правила маршрутизации flask. Любые переменные url будут переданы методу `resource` в качестве `args`.
- `endpoint` (строка) – имя конечной точки (по умолчанию `Resource.__name__.lower()`) Может использоваться для ссылки на этот маршрут в полях `fields.Url`
- `resource_class_args` (кортеж) – аргументы (`args`), которые будут переданы в конструктор ресурса.
- `resource_class_kwargs` (словарь) – аргументы (`kwargs`), которые будут переданы конструктору ресурса.

Метод `run()` запускает приложение на локальном сервере разработки. В качестве параметров, данный метод может принимать в себя:

- `host` (`Optional[str]`) - имя хоста для прослушивания. Можно использовать значение `'0.0.0.0'`, чтобы сервер был доступен и для внешнего использования. По умолчанию установлено значение `'127.0.0.1'` или имя хоста в конфигурационной переменной `SERVER_NAME`, если оно присутствует.
- `port` (`Optional[int]`) - порт веб-сервера. По умолчанию 5000 или порт, определенный в конфигурационной переменной `SERVER_NAME`, если он присутствует.
- `debug` (`Optional[bool]`) - если задано, включает или выключает режим отладки.

- `load_dotenv (bool)` - загрузка ближайших файлы `.env` и `.flaskenv` для установки переменных окружения. Также изменит рабочий каталог на каталог, содержащий первый найденный файл.
- `options (Any)` - опции, которые должны быть переданы базовому серверу Werkzeug.

Поскольку первым параметром в методе `add_resource()` передается имя класса ресурса, был создан класс `Documents`, в котором реализованы основные методы для HTTP запросов:

- Метод `get()`. Возвращает весь список документов, которые в данный момент находятся в базе данных. Также можно просмотреть только один документ, указав его `id`. Если по указанному `id` не найдется ни одного документа, запрос выдаст ошибку «File not found» (Листинг 15):

Листинг 15. Реализация метода `get()`

```
def get(self, id=0):
    files_enum(Path(__file__).parent/'university-documents')
    files_list = get_database_data()

    if id == 0:
        return files_list, 200

    for file in files_list:
        if file['id'] == id:
            return file, 200

    return 'File not found', 400
```

- Метод `post()`. При обращении можно добавить новый файл. При попытке добавить существующий файл API выдаст ошибку (Листинг 16):

Листинг 16. Реализация метода `post()`

```
def post(self, id):
    files_enum(Path(__file__).parent/'university-documents')
    current_files = get_database_data()
    parser = reqparse.RequestParser()
    parser.add_argument("name")
    parser.add_argument("date")
    parser.add_argument("visible")
    params = parser.parse_args()

    for file in current_files:
        if (id == file["id"]):
            return f"File with id {id} already exists", 400

    file = {
        "id": int(id),
        "name": params["name"],
        "date": params["date"],
        "visible": params["visible"]
    }

    current_files.append(file)
    return file, 201
```

- Метод `delete()`. При обращении удаляет файл по указанному `id` (Листинг 17):

Листинг 17. Реализация метода `delete()`

```
def delete(self, id):
    files_enum(Path(__file__).parent/'university-
documents')
    current_files = get_database_data()
    current_files = [file for file in current_files if
file["id"] != id]
    return f"File with id {id} is deleted.", 200
```

В каждом из методов мы обращаемся к базе данных, используя функцию `get_database_data()`, чтобы получить весь список документов в формате список объектов.

После отправки изменений в облачную платформу Heroku, достаточно зайти на выделенный домен, чтобы увидеть результат работы API (рис. 9).

```
[
  {
    "id": 31,
    "name": "Error404_2.docx",
    "date": "",
    "visible": 1,
    "email": "rahilova@sfedu.ru",
    "telegramId": "248936609"
  },
  {
    "id": 32,
    "name": "links.docx",
    "date": "",
    "visible": 1,
    "email": "rahilova@sfedu.ru",
    "telegramId": "248936609"
  },
  {
    "id": 34,
    "name": "test3.docx",
    "date": "",
    "visible": 1,
    "email": "rahilova@sfedu.ru",
    "telegramId": "248936609"
  },
  {
    "id": 42,
    "name": "otchet.docx",
    "date": "05.06.2022",
    "visible": 1,
    "email": "",
    "telegramId": ""
  }
]
```

Рис. 9. Результат работы API

2.6 Тестирование созданного сервиса в чат-боте для контроля сроков сдачи документов

Для тестирования созданного API, был реализован чат-бот в приложении «Телеграмм», уведомляющий пользователя об окончании срока сдачи документа.

Чат-боты кардинально меняют способы взаимодействия людей с технологиями. В последние годы их простота и минимальные расходы способствовали их внедрению в различных областях и отраслях.

Существует множество вариантов того, где можно разместить чатбота, и одним из наиболее распространенных вариантов являются платформы социальных сетей, поскольку большинство людей регулярно ими пользуются.

Telegram — одна из самых популярных IM-платформ сегодня, поскольку она позволяет хранить сообщения в облаке, а не только на устройстве пользователя, и может предложить хорошую мультиплатформенную поддержку так как Telegram можно использовать на Android, iOS, Windows и практически на любой другой платформе, поддерживающей веб-версию.

Для разработки чат-бота также использовался язык программирования Python.

Чтобы создать чат-бота в Telegram, необходимо обратиться к BotFather, который, по сути, является ботом, используемым для создания других ботов (рис. 10).

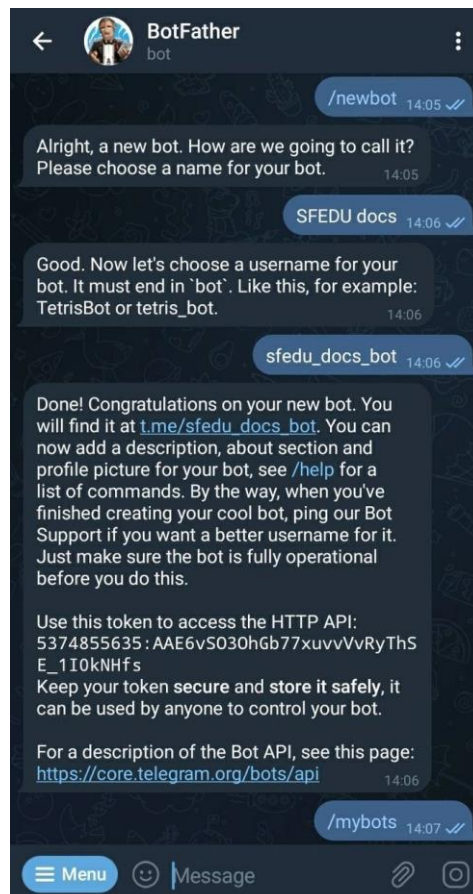


Рис. 10. Формирование токена для будущего чат-бота

Рядом с программой для сканирования документов создадим файл «alert.py», который будет запускать чат-бота.

Для доступа к данным из API создадим функцию `get_api_data()`, которая будет возвращать данные в формате JSON (Листинг 18).

Листинг 18. Функция `get_api_data()` для доступа к данным API

```
def get_api_data():
    response_API = requests.get('https://sfedu-document-
service.herokuapp.com/document-service')
    data = response_API.text
    return json.loads(data)
```

Для отправки уведомлений об окончании срока сдачи документа опишем функцию `send_alert()` (Листинг 19).

Листинг 19. Функция send_alert() для отправки уведомления в чат-бот

```
def send_alert(user_id, text):  
    token = '5374855635:AAE6vSO3OhGb77xuvv  
VvRyThSE_1I0kNHfs'  
    requests.get(f'https://api.telegram.org/bot{token}  
/sendMessage?chat_id="{user_id}"&text="{text}"')
```

Перечислим входные данные функции:

- user_id — id пользователя в приложении Telegram;
- text — текст, отображаемый в уведомлении в чат-боте;

Также, в данной функции использован токен доступа, сформированный ранее в телеграм-боте «BotFather».

Далее, при вызове программы мы создаем переменную «current_timestamp», которая содержит текущую дату и время, чтобы в дальнейшем сравнивать ее с датой полученной из API. Также учитываем рабочий день пользователя (по умолчанию с 9:00 – 18:00), поскольку именно это время будут приходить уведомления (по умолчанию каждый час). Данное условие можно настроить под каждого пользователя отдельно, поскольку рабочее время у всех разное.

Также обработан случай, когда срока сдачи нет в документе. При срабатывании данного условия, документу автоматически устанавливается срок сдачи равный шести дням (Листинг 20).

Листинг 20. Сравнение даты, взятой из API и сегодняшней

```
while True:  
    current_timestamp =  
int(datetime.timestamp(datetime.now()))  
    if 32400 <= current_timestamp % 86400 < 64800:  
        if not current_timestamp % 3600:  
            documents = get_api_data()  
            for document in documents:
```

```

        name = document['name']
        date = document['date']
        user = document['telegramId']
        try:
            datetime_variable = datetime.strptime(date,
'%d.%M.%Y')
        except:
            date = str(current_timestamp + 86400*6)

```

Если до срока сдачи документа больше суток, но меньше недели, оповещение от бота будет поступать каждый день примерно в 9 утра (Листинг 21)

Листинг 21. Срок сдачи документа больше суток, но меньше недели

```

document_timestamp = int(datetime.timestamp(datetime_variable))
if 86400 < document_timestamp - current_timestamp < 86400*7 and
current_timestamp % 86400 <= 33000:
    message = f'Обратите внимание, на срок сдачи документа
{name}!\nПоследняя неделя! Сдать до {date}!'
    send_alert(user, message)

```

Если до срока сдачи меньше суток, оповещение от бота будет поступать каждый час в течение дня (Листинг 22)

Листинг 22. Срок сдачи документа меньше суток

```

elif document_timestamp - current_timestamp < 86400:
    message = f'Обратите внимание, на срок сдачи документа
{name}!\nСегодня {date}! День сдачи!'
    send_alert(user, message)

```

Заключение

В рамках данной выпускной квалификационной работы был проведен анализ в разработке API, а также создан сервис, содержащий данные с полезной информацией необходимой пользователю для отображения в программах планирования задач. Созданный сервис опубликован на облачной платформе Heroku.

Задачи, которые стояли, были осуществлены, а именно:

- Анализ функционала API;
- Реализация программы на языке программирования Python, собирающую данные о документах из рабочей папки на компьютере;
- Настройка облачной платформы Heroku, для публикации API;
- Подключение реализованной программы к базе данных MySQL;
- Реализация API, содержащее данные с полезной информацией необходимой пользователю для отображения в программах планирования задач;
- Публикация готового сервиса по анализу документов на облачную платформу Heroku;
- Тестирование сервиса в созданном чат-боте по отслеживанию сроков сдачи документов;

Результаты данной работы были доложены на конференции в рамках Неделя науки 2022 Института математики, механики и компьютерных наук им И.И. Воровича ЮФУ 12 мая 2022 г и рекомендованы к публикации (в настоящее время сборник трудов оформляется).

Ссылка на GitHub: <https://github.com/PiuPiPiu/document-service>

Список литературы

1. Python documentation. [Электронный ресурс] — 2022 — режим доступа: <https://docs.python.org/3/> (Дата обращения 17.08.2021)
2. What is an API. [Электронный ресурс] — 2022— режим доступа: <https://www.mulesoft.com/resources/api/what-is-an-api> (Дата обращения 20.10.2021)
3. What is API: Definition, Types, Specifications, Documentation. [Электронный ресурс] — 2021 — режим доступа: <https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/> (Дата обращения 22.11.2021)
4. Documentation for the Heroku platform. [Электронный ресурс] — 2022 — режим доступа: <https://devcenter.heroku.com/categories/reference> (Дата обращения 13.01.2022)
5. Flask documentation [Электронный ресурс] — 2022 — режим доступа: <https://flask.palletsprojects.com/en/2.1.x/> (Дата обращения 25.01.2022)
6. Flask RESTful documentation [Электронный ресурс] — 2022 — режим доступа: <https://flask-restful.readthedocs.io/en/latest/> (Дата обращения 12.02.2022)
7. MySQL documentation [Электронный ресурс] — 2022 — режим доступа: <https://dev.mysql.com/doc/> (Дата обращения 27.02.2022)
8. JawsDB documentation [Электронный ресурс] — 2022 — режим доступа: <https://www.jawsdb.com/docs/#jawsdb> (Дата обращения 28.02.2022)
9. Telegram bot API [Электронный ресурс] — 2022 — режим доступа: <https://core.telegram.org/bots/api> (Дата обращения 07.03.2022)
10. Telegram bots book [Электронный ресурс] — 2022 — режим доступа: <https://telegrambots.github.io/book/> (Дата обращения 14.03.2022)